

期末复习

苏州大学 计算机科学与技术学院

汪笑宇

Email: xywang21@suda.edu.cn

渐近表示法

■ $f(n)=\Theta(g(n))$ 渐近紧确界

$\Theta(g(n)) = \{f(n): \text{存在正常量 } c_1, c_2, n_0, \text{ 使得对所有 } n \geq n_0 \text{ 有 } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

■ $f(n)=O(g(n))$ 渐近上界

$O(g(n)) = \{f(n): \text{存在正常量 } c, n_0, \text{ 使得对所有 } n \geq n_0 \text{ 有 } 0 \leq f(n) \leq c g(n)\}$

■ $f(n)=\Omega(g(n))$ 渐近下界

$\Omega(g(n)) = \{f(n): \text{存在正常量 } c, n_0, \text{ 使得对所有 } n \geq n_0 \text{ 有 } 0 \leq c g(n) \leq f(n)\}$

渐近表示法 (续)

■ $f(n)=o(g(n))$ 渐近非紧上界

$o(g(n)) = \{f(n): \text{对任意常数 } c>0, \text{ 存在常数 } n_0 > 0, \text{ 使得对所有 } n \geq n_0 \text{ 有 } 0 \leq f(n) < cg(n)\}$

■ $f(n)=\omega(g(n))$ 渐近非紧下界

$\omega(g(n)) = \{f(n): \text{对任意常数 } c>0, \text{ 存在常数 } n_0 > 0, \text{ 使得对所有 } n \geq n_0 \text{ 有 } 0 \leq cg(n) < f(n)\}$

$$\blacksquare \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f(n) = o(g(n)), f(n) = O(g(n)) \\ a, a > 0 & f(n) = \Theta(g(n)), f(n) = O(g(n)), f(n) = \Omega(g(n)) \\ \infty & f(n) = \omega(g(n)), f(n) = \Omega(g(n)) \end{cases}$$

递归式与分治法

■求解递归式方法（得出算法 Θ 或 O 渐近界）：

- **代入法**：猜测一个界，然后用数学归纳法证明这个界是正确的
- **递归树法**：将递归式转换为一棵树，结点表示不同层次的递归调用产生的代价，然后采用边界和技术来求解递归式
- **主方法**：可求解形如 $T(n)=aT(n/b)+f(n)$ 递归式的界

分治法求解——最大子数组问题

- （教材p39）寻找数组 $A[1..n]$ 中，和最大的非空连续子数组

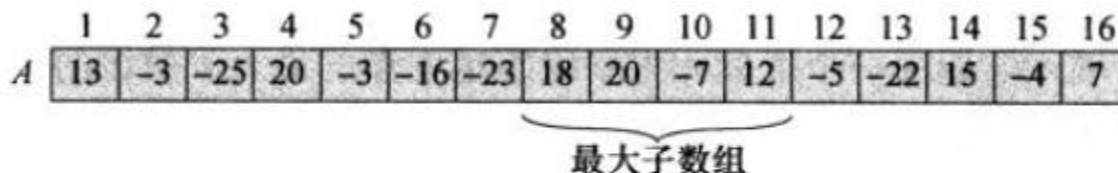


图 4-3 股票价格变化值的最大子数组问题。本例中，子数组 $A[8..11]$ 的和是 43，是 A 的所有连续子数组中和最大的

- 暴力求解：穷举所有非空连续子数组，即穷举所有可能的起始、终止位置： $C_n^2 + \boxed{n} = \frac{n(n-1)}{2} + n = \Theta(n^2)$
起始终止位置相同
- 可在 $O(n^3)$ 时间解决，改进方法可在 $O(n^2)$ 时间解决

分治法求解——最大子数组问题 (续)

■分治法适用条件分析：

➤该问题的规模缩小到一定程度就可容易解决 ✓

- 分析：如果 $n=1$ 即只有一个元素，只要输出该元素值即可

➤该问题可以分解为若干个规模较小的相同问题 ✓

- 分析：将数组 $A[low..high]$ 对半划分，中间位置为 mid ， A 的非空连续子数组 $A[i..j]$ 肯定在三种情况之一：

1. 完全位于子数组 $A[low..mid]$ 中： $low \leq i \leq j \leq mid$
2. 完全位于子数组 $A[mid+1..high]$ 中： $mid+1 \leq i \leq j \leq high$
3. 跨越了中点 mid ： $low \leq i \leq mid < j \leq high$

前两种情况相当于求解规模更小的原问题；第3种情况易于计算

分治法求解——最大子数组问题 (续)

■分治法适用条件分析：

➤分解出的子问题的解可以合并为原问题的解 ✓

- 分析：将三种情况得到的最大和进行比较即可

➤分解出的各个子问题是相互独立的 ✗

- 分析：求和过程可能多次计算，并非独立

存在更好的解法：动态规划

分治法求解——最大子数组问题 (续)

■ 寻找跨越中点 mid 的最大子数组

$A[\max_left..\max_right] : low \leq \max_left \leq mid < \max_right \leq high$

FIND_MAX_CROSSING_SUBARRAY($A, low, mid, high$)

1 $left_sum \leftarrow -\infty$

2 $sum \leftarrow 0$

3 **for** $i \leftarrow mid$ **downto** low **do**

4 $sum \leftarrow sum + A[i]$

5 **if** $sum > left_sum$

6 $left_sum \leftarrow sum; \max_left \leftarrow i$

7 $right_sum \leftarrow -\infty$

8 $sum \leftarrow 0$

9 **for** $j \leftarrow mid+1$ **to** $high$ **do**

10 $sum \leftarrow sum + A[j]$

11 **if** $sum > right_sum$

12 $right_sum \leftarrow sum; \max_right \leftarrow j$

13 **return** ($\max_left, \max_right, left_sum + right_sum$)

$\Theta(n)$

分治法求解——最大子数组问题 (续)

```
FIND_MAXIMUM_SUBARRAY(A, low, high)
1  if high = low
2    return (low, high, A[low])
3  else mid  $\leftarrow \lfloor (\textit{low} + \textit{high})/2 \rfloor$ 
4    (left_low, left_high, left_sum)  $\leftarrow$ 
      FIND_MAXIMUM_SUBARRAY(A, low, mid) // $T(n/2)$ 
5    (right_low, right_high, right_sum)  $\leftarrow$ 
      FIND_MAXIMUM_SUBARRAY(A, mid+1, high) // $T(n/2)$ 
6    (cross_low, cross_high, cross_sum)  $\leftarrow$ 
      FIND_MAX_CROSSING_SUBARRAY(A, low, mid, high) // $\Theta(n)$ 
7  if left_sum  $\geq$  right_sum and left_sum  $\geq$  cross_sum
8    return (left_low, left_high, left_sum)
9  elseif right_sum  $\geq$  left_sum and right_sum  $\geq$  cross_sum
10   return (right_low, right_high, right_sum)
11 else return (cross_low, cross_high, cross_sum)
```

分治法求解——最大子数组问题 (续)

■ 算法分析

$$T(n) = 2T(n/2) + \Theta(n)$$

➤ 与归并排序类似

➤ 主方法求解： $a=2$, $b=2$, $f(n)=cn$, $n^{\log_b a} = n^{\log_2 2} = \Theta(n)$

➤ $f(n) = \Theta(n)$

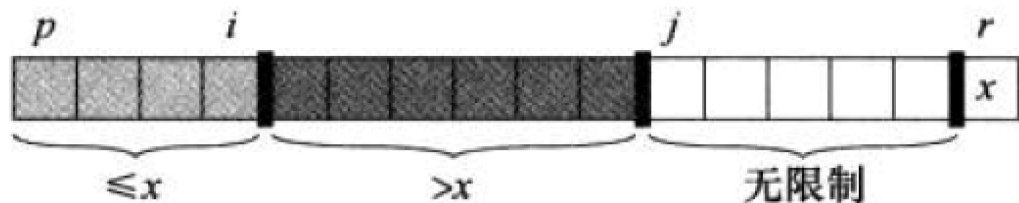
➤ $n^{\log_b a}$ 与 $f(n)$ 同级别，主定理第2种情况： $T(n)=\Theta(n \lg n)$

分治法求解——快速排序

■ 算法描述

QUICKSORT(A, p, r)

```
1 if  $p < r$ 
2    $q \leftarrow \text{PARTITION}(A, p, r)$ 
3   QUICKSORT( $A, p, q-1$ )
4   QUICKSORT( $A, q+1, r$ )
```



PARTITION(A, p, r)

```
1  $x \leftarrow A[r]$  //划分元/主元(pivot element)
2  $i \leftarrow p - 1$ 
3 for  $j \leftarrow p$  to  $r-1$  do
4   if  $A[j] \leq x$ 
5      $i \leftarrow i + 1$ , exchange  $A[i]$  with  $A[j]$ 
6 exchange  $A[i+1]$  with  $A[r]$ 
7 return  $i+1$ 
```

循环不变式:

1. $A[p..i] \leq x$
2. $A[i+1..j-1] > x$
3. $A[r] = x$

该划分使得:

$$A[p..q-1] \leq A[q] < A[q+1..r]$$

分治法求解——快速排序 (续)

■随机化版本（教材p100）

- 快速排序的平均性能假定：输入的所有排列是等可能的
- 算法随机化是指：算法行为不仅由输入确定，而且与随机数发生器产生的值有关，强迫输入分布是随机的

```
RANDOMIZED_PARTITION( $A, p, r$ )  
1  $i \leftarrow \text{RANDOM}(p, r)$   
2 exchange  $A[r]$  with  $A[i]$   
3 return PARTITION( $A, p, r$ )
```

- 分析较困难
- 算法非常有效，排序过程中，某次随机选择最坏不会影响总体效果

Sherwood算法实例——快速排序

RAND_QUICKSORT($A, low, high$)

```
1  //A: 待排序数组, low/high: 排序起始/终止下标
2  if  $low < high$ 
3       $i \leftarrow \text{RANDOM}(low, high)$ ; //low..high随机抽取一个下标
4      swap( $A[high]$ ,  $A[i]$ )
5       $k \leftarrow \text{PARTITION}(A, low, high)$ 
6      RAND_QUICKSORT( $A, low, k-1$ )
7      RAND_QUICKSORT( $A, k+1, high$ )
```

引入随机因素

SHUFFLE(A)

```
1   $n \leftarrow A.length$ 
2  for  $i \leftarrow 1$  to  $n-1$  do
3      //在 $A[i..n]$ 中随机选一个元素放在 $A[i]$ 上
4       $j \leftarrow \text{RANDOM}(i, n)$ 
5      swap( $A[i]$ ,  $A[j]$ )
6  执行原确定性算法
```

原算法较复杂，很难对其进行修改时可适用

输入实例随机处理

矩阵链乘法

- 给定 n 个矩阵的序列（矩阵链） $\langle A_1, A_2, \dots, A_n \rangle$ ，
计算乘积 $A_1 A_2 \cdots A_n$
- 计算多个矩阵连乘积可用**括号**来决定计算次序，
每一个括号内的矩阵相乘调用**标准的矩阵乘法**
- 矩阵积的完全括号化
 - 它是单一矩阵
 - 或者是两个完全括号化的矩阵链的积

递归定义
计算次序无二义性

$$(A_1 (A_2 (A_3 A_4)))$$

$$((A_1 (A_2 A_3)) A_4)$$

$$(A_1 ((A_2 A_3) A_4))$$

$$(((A_1 A_2) A_3) A_4)$$

$$((A_1 A_2) (A_3 A_4))$$

矩阵链乘法 (续)

■不同的括号化方式产生不同的计算成本

```
MATRIX_MULTIPLY(A, B)
1  if  $A.columns \neq B.rows$ 
2    error “incompatible dimensions”
3  else let C be a new  $A.rows \times B.columns$  matrix
4    for  $i \leftarrow 1$  to  $A.rows$  do
5      for  $j \leftarrow 1$  to  $B.columns$  do
6         $c_{ij} \leftarrow 0$ 
7        for  $k \leftarrow 1$  to  $A.columns$  do
8           $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$ 
9  return C
```

- 第8行执行次数： $A.rows \times B.columns \times A.columns$ (或 $B.rows$)
- 设A是 $p \times q$ 矩阵、B是 $q \times r$ 矩阵，则计算 $C=A \cdot B$ 共需 pqr 次标量乘法

矩阵链乘法 (续)

■ 矩阵链乘法问题实质上是一个**最优括号化**问题：

- 给定 n 个矩阵的链 $\langle A_1, A_2, \dots, A_n \rangle$ ，矩阵 A_i 的规模为 $p_{i-1} \times p_i$ ($1 \leq i \leq n$)，求**完全括号化方案**，使得计算乘积 $A_1 A_2 \cdots A_n$ 所需**标量乘法次数最少**
- 计算括号化方案数量：设 $P(n)$ 表示一个 n 个矩阵的链中**可选括号化方案数量**，则穷举法产生数量：

$$P(n) = \begin{cases} 1, & n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k), & n \geq 2. \end{cases}$$

Catalan数，指数阶： $\Omega(4^n/n^{1.5})$ ，不如直接求解矩阵乘积！

矩阵链乘法 (续)

■刻画一个最优解的结构特征——最优括号化方案的结构特征

➤ $A_{i..j}$ ($1 \leq i \leq j \leq n$): $A_i A_{i+1} \cdots A_j$

➤ 设 $A_i A_{i+1} \cdots A_j$ 的最优括号化是在 A_k 和 A_{k+1} 之间划分开 ($i \leq k < j$ 且 $i < j$)

➤ 对某个 k , 先计算 $A_{i..k}$ 和 $A_{k+1..j}$, 再计算两者乘积得到 $A_{i..j}$

➤ 计算代价:

$A_{i..k}$ 计算代价 + $A_{k+1..j}$ 计算代价 + 两者乘积计算代价

矩阵链乘法 (续)

- 关键： $A_i A_{i+1} \cdots A_j$ 的最优括号化亦要求分割开的两个子链 $A_{i..k}$ 和 $A_{k+1..j}$ 是最优括号化。
- 可用反证法证明：若 $A_{i..k}$ 括号化不是最优，则可找到一个成本更小的方法将其括号化，代入到 $A_{i..j}$ 的最优括号化表示中，得到的计算成本比最优解小，矛盾！

矩阵链乘法 (续)

■ $m[i, j]$: 计算 $A_{i..j}$ 所需标量乘法次数的最小值

➤ 若 $i < j$, 利用步骤1最优子结构计算代价 (k 是最优分割点) :

$A_{i..k}$ 计算代价 + $A_{k+1..j}$ 计算代价 + 两者乘积计算代价
即: $m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$

➤ 以上公式成立需要 k 是最优分割点

➤ 检查所有 $j-i$ 种可能的 k 即可:

$$m[i, j] = \begin{cases} 0, & i = j, \\ \min_{i \leq k < j} \{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\}, & i < j. \end{cases}$$

➤ 定义 $s[i, j]$ 保存 $A_{i..j}$ 最优括号化方案分割点位置 k

矩阵链乘法 (续)

MATRIX_CHAIN_ORDER(p)

```
1   $n \leftarrow p.length - 1$ 
2  let  $m[1..n, 1..n]$  and  $s[1..n-1, 2..n]$  be new tables
3  for  $i \leftarrow 1$  to  $n$  do
4       $m[i, i] \leftarrow 0$  //  $i = j$ 
5  for  $l \leftarrow 2$  to  $n$  do //  $l$ : 矩阵链长度,  $i \neq j$  时  $1 \leq j-i = l-1 \leq n-1$ 
6      for  $i \leftarrow 1$  to  $n-l+1$  do
7           $j \leftarrow i + l - 1$ 
8           $m[i, j] \leftarrow \infty$ 
9          for  $k \leftarrow i$  to  $j-1$  do
10              $q \leftarrow m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$ 
11             if  $q < m[i, j]$ 
12                  $m[i, j] \leftarrow q$ 
13                  $s[i, j] \leftarrow k$ 
14 return  $m$  and  $s$ 
```

矩阵链乘法 (续)

- 时间复杂度： $O(n^3)$ ，三层循环
- 空间复杂度： $O(n^2)$ ，保存表 m 和 s
- 较穷举方法指数阶高效得多

矩阵链乘法 (续)

■构造最优解

- MATRIX_CHAIN_ORDER求出了计算矩阵链乘积所需的最少标量乘法次数，但并未指出如何进行这种最优代价矩阵链乘法计算
- 表 s 记录了构造最优解的最优分割信息，可递归求出其中最外层划分位置： $k = s[1, n]$ ，则进一步求 $s[1, k]$ 和 $s[k+1, n]$ ，直到 $s[i, j]$ 中 $i=j$ 为止

```
PRINT_OPTIMAL_PARENS( $s, i, j$ )
1  if  $i = j$ 
2    print " $A_i$ "
3  else print "("
4    PRINT_OPTIMAL_PARENS( $s, i, s[i, j]$ )
5    PRINT_OPTIMAL_PARENS( $s, s[i, j]+1, j$ )
6    print ")"
```

矩阵链乘法 (续)

■按照最优括号化计算矩阵链乘积

```
MATRIX_CHAIN_MULTIPLY( $\mathcal{A}$ ,  $s$ ,  $i$ ,  $j$ )  
1  if  $i = j$   
2    return  $A_i$   
3  else  
4     $X \leftarrow \text{MATRIX\_CHAIN\_MULTIPLY}(\mathcal{A}, s, i, s[i, j])$   
5     $Y \leftarrow \text{MATRIX\_CHAIN\_MULTIPLY}(\mathcal{A}, s, s[i, j]+1, j)$   
6    return  $\text{MATRIX\_MULTIPLY}(X, Y)$ 
```

最长公共子序列LCS

■子序列：将给定序列中零个或多个元素去掉之后得到的结果

➤形式化定义：给定一个序列 $X=\langle x_1, x_2, \dots, x_m \rangle$ ，另一个序列 $Z=\langle z_1, z_2, \dots, z_k \rangle$ 满足如下条件时称为 X 的子序列：存在一个严格递增的 X 的下标序列 $\langle i_1, i_2, \dots, i_k \rangle$ ，对所有 $j=1, 2, \dots, k$ ，满足 $x_{i_j}=z_j$

➤例： $X=\langle A, B, C, B, D, A, B \rangle$

$Z=\langle B, C, D, B \rangle$ 为 X 的子序列，对应下标序列为 $\langle 2, 3, 5, 7 \rangle$

子序列不一定是由原序列连续元素构成的序列！

最长公共子序列LCS (续)

- 公共子序列 (common subsequence): 给定两个序列 X 和 Y , 如果 Z 既是 X 的子序列, 也是 Y 的子序列, 则称 Z 是 X 和 Y 的公共子序列
- 最长公共子序列问题 (longest-common-subsequence problem): 求两个序列公共子序列中最长的一个
- 求解两个给定序列的LCS
 1. 刻画LCS结构特征
 2. 递归解
 3. 计算LCS长度
 4. 构造LCS

最长公共子序列LCS (续)

1. 刻画LCS特征

- 穷举法：穷举 X 的所有子序列，检查是否也是 Y 的子序列。若 $|X|=m$ ，则子序列共 2^m 个，**穷举法为指数阶下界**
- LCS具有最优子结构性质
前缀：给定一个序列 $X=\langle x_1, x_2, \dots, x_m \rangle$ ，对 $i=0, 1, \dots, m$ ，定义 X 的第 i 前缀为 $X_i=\langle x_1, x_2, \dots, x_i \rangle$
- **定理15.1** 令 $X=\langle x_1, x_2, \dots, x_m \rangle$ 和 $Y=\langle y_1, y_2, \dots, y_n \rangle$ 为两个序列， $Z=\langle z_1, z_2, \dots, z_k \rangle$ 为 X 和 Y 的任意LCS
 1. 若 $x_m=y_n$ ，则 $z_k=x_m=y_n$ 且 Z_{k-1} 是 X_{m-1} 和 Y_{n-1} 的一个LCS；
 2. 若 $x_m \neq y_n$ ，则 $z_k \neq x_m$ 意味着 Z 是 X_{m-1} 和 Y 的一个LCS；
 3. 若 $x_m \neq y_n$ ，则 $z_k \neq y_n$ 意味着 Z 是 X 和 Y_{n-1} 的一个LCS。

最长公共子序列LCS (续)

➤定理15.1 证明（反证法）：

1. (1) $z_k = x_m = y_n$ ：若 $z_k \neq x_m$ ，则可将 $x_m = y_n$ 追加到 Z 的末尾，得到 X 和 Y 的一个长度为 $k+1$ 的公共子序列，矛盾！

(2) Z_{k-1} 是 X_{m-1} 和 Y_{n-1} 的一个 LCS：若 X_{m-1} 和 Y_{n-1} 存在长度大于 $k-1$ 的公共子序列 W ，则可将 $x_m = y_n$ 追加到 W 末尾，得到 X 和 Y 的一个长度大于 k 的公共子序列，矛盾！

2. 因为 $x_m \neq y_n$ ，所以 X_{m-1} 和 Y 的 LCS 与 X 和 Y 的 LCS 相同。若存在 X_{m-1} 和 Y 长度大于 k 的公共子序列 W ，则 W 也是 X 和 Y 的公共子序列，长度大于 k ，矛盾！

3. 与2对称

最长公共子序列LCS (续)

2. 递归解

- 由定理15.1：求 $X=\langle x_1, x_2, \dots, x_m \rangle$ 和 $Y=\langle y_1, y_2, \dots, y_n \rangle$ 的一个LCS时，需求解一个或两个子问题：
- 若 $x_m=y_n$ ，则求解 X_{m-1} 和 Y_{n-1} 的一个LCS，将 $x_m=y_n$ 追加到这个LCS的末尾
 - 若 $x_m \neq y_n$ ，(1) 求解 X_{m-1} 和 Y 的LCS；(2) 求解 X 和 Y_{n-1} 的LCS。求两者长度最大者

最长公共子序列LCS (续)

2. 递归解

➤ $c[i, j]$: X_i 和 Y_j 的LCS长度 ($0 \leq i \leq m, 0 \leq j \leq n$)

$$c[i, j] = \begin{cases} 0, & i = 0 \text{ or } j = 0, \\ c[i - 1, j - 1] + 1, & i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i, j - 1], c[i - 1, j]), & i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

➤ 限定了需求解哪些子问题，并非所有子问题都要求解

最长公共子序列LCS (续)

3. 计算LCS长度

➤不同子问题个数： $\Theta(mn)$

➤输入：序列 $X=\langle x_1, x_2, \dots, x_m \rangle$ 和 $Y=\langle y_1, y_2, \dots, y_n \rangle$

➤输出： $c[0..m, 0..n]$ ：按行主次序计算LCS长度
 $b[1..m, 1..n]$ ：辅助构造最优解子序列

$$b[i, j] = \begin{cases} \nwarrow, & c[i, j] = c[i-1, j-1] + 1, \\ \uparrow, & c[i, j] = c[i-1, j], \\ \leftarrow, & c[i, j] = c[i, j-1]. \end{cases}$$

➤构造解时，从 $b[m, n]$ 出发，根据箭头方向上溯至 $i=0$ 或 $j=0$ 为止，当 $b[i, j]$ 包含“ \nwarrow ”时打印出 x_i 即可

最长公共子序列LCS (续)

LCS_LENGTH(X, Y)

```
1   $m \leftarrow X.length$ 
2   $n \leftarrow Y.length$ 
3  let  $b[1..m, 1..n]$  and  $c[0..m, 0..n]$  be new tables
4  for  $i \leftarrow 1$  to  $m$  do
5       $c[i, 0] \leftarrow 0$ 
6  for  $j \leftarrow 0$  to  $n$  do
7       $c[0, j] \leftarrow 0$ 
8  for  $i \leftarrow 1$  to  $m$  do    // 依次考虑 $X_1, X_2, \dots, X_m$ 的前缀子列
9      for  $j \leftarrow 1$  to  $n$  do // 依次考虑 $Y_1, Y_2, \dots, Y_n$ 的前缀子列
10         if  $x_i = y_j$         // 两个前缀子列的最后一位相同
11              $c[i, j] \leftarrow c[i-1, j-1] + 1$ 
12              $b[i, j] \leftarrow \text{“}\nwarrow\text{”}$ 
13         elseif  $c[i-1, j] \geq c[i, j-1]$  // 两个前缀子列的最后一位不同
14              $c[i, j] \leftarrow c[i-1, j]$ 
15              $b[i, j] \leftarrow \text{“}\uparrow\text{”}$ 
16         else  $c[i, j] \leftarrow c[i, j-1]$ 
17              $b[i, j] \leftarrow \text{“}\leftarrow\text{”}$ 
18 return  $c$  and  $b$ 
```

$\Theta(mn)$

最长公共子序列LCS (续)

		j	0	1	2	3	4	5	6	
				y_j	B	D	C	A	B	A
i	x_i									
0	x_i		0	0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1	
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2	
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2	
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3	
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3	
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4	
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4	

$X = \langle A, B, C, B, D, A, B \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

最长公共子序列LCS (续)

4. 构造LCS

- 从 $b[m, n]$ 开始根据箭头上溯至 $i=0$ 或 $j=0$ 即可
 - 当 $b[i, j] = \text{“}\nwarrow\text{”}$ 时, 有 $x_i = y_j$ 是LCS的一个元素
 - 逆序构造出LCS, 可用递归算法顺序打印

```
PRINT_LCS( $b, X, i, j$ )  
1  if  $i = 0$  or  $j = 0$   
2    return  
3  if  $b[i, j] = \text{“}\nwarrow\text{”}$   
4    PRINT_LCS( $b, X, i-1, j-1$ )  
5    print  $x_i$   
6  elseif  $b[i, j] = \text{“}\uparrow\text{”}$   
7    PRINT_LCS( $b, X, i-1, j$ )  
8  else PRINT_LCS( $b, X, i, j-1$ )
```

$O(m+n)$

每次递归调用
 i 和 j 至少一个会减少1

活动选择问题

- 多个活动竞争资源的调度问题：尽可能多地选择互不冲突的活动
- 设有 n 个活动（activity） $S=\{a_1, a_2, \dots, a_n\}$ ，均要使用某资源（如教室），该资源使用方式为独占式，一次只供一个活动使用
 - 每个活动 a_i 发生的时间为 $[s_i, f_i)$, $0 \leq s_i < f_i < \infty$
 - 两活动 a_i, a_j 兼容（compatible不冲突）： $[s_i, f_i), [s_j, f_j)$ 不重叠，满足 $s_i \geq f_j$ 或 $s_j \geq f_i$ ，即：一活动的开始时间大于等于另一活动的完成时间
 - 活动选择问题：选择最多的互不冲突的活动，使兼容活动集合最大，即求解 $A \subseteq S$ ， A 中活动互不冲突且 $|A|$ 最大

贪心算法——活动选择问题

3. 贪心算法

- 直观上，我们应该选择这样一个活动，选出它后剩下的资源应能被尽量多的其他任务所用
- 每次选择候选集中**最早结束的活动**
- **定理16.1** 考虑任意非空子问题 S_{ij} ，令 a_m 是 S_{ij} 中结束时间最早的活动，即： $f_m = \min\{f_k: a_k \in S_{ij}\}$ ，则
 1. a_m 在 S_{ij} 的某个最大兼容活动子集中
 2. 子问题 S_{im} 的解是空集

活动选择问题 (续)

3. 贪心算法

➤ **定理16.1** 考虑任意非空子问题 S_{ij} , 令 a_m 是 S_{ij} 中结束时间最早的活动, 即: $f_m = \min\{f_k: a_k \in S_{ij}\}$, 则

1. a_m 在 S_{ij} 的某个最大兼容活动子集中
2. 子问题 S_{im} 的解是空集

➤ **证明:** (第2部分, 反证法) 假定 S_{im} 的解非空, 则存在 $a_k \in S_{im}$, 使得 $f_i \leq s_k < f_k \leq s_m$ 。由此得到 $a_k \in S_{ij}$ 的完成时间先于 a_m , 与 a_m 是 S_{ij} 最早完成的活动矛盾

活动选择问题 (续)

3. 贪心算法

➤ **定理16.1** 考虑任意非空子问题 S_{ij} ，令 a_m 是 S_{ij} 中结束时间最早的活动，即： $f_m = \min\{f_k: a_k \in S_{ij}\}$ ，则

1. a_m 在 S_{ij} 的某个最大兼容活动子集中
2. 子问题 S_{im} 的解是空集

➤ **证明：**（第1部分）设 A_{ij} 是 S_{ij} 的某个最优解，假设 A_{ij} 中的活动已按完成时间单调递增排序，且 a_k 是 A_{ij} 中最早结束的活动：

- 1、若 $a_k = a_m$ ，则问题已得证，即最优解包含 a_m ；
- 2、若 $a_k \neq a_m$ ，构造子集 $A'_{ij} = (A_{ij} - \{a_k\}) \cup \{a_m\}$ ，即将最优解中的 a_k 替换为 a_m ，则需证明 A'_{ij} 也是最优解
因为 $f_m \leq f_k$ ，因此 A'_{ij} 中的活动也不冲突，且 $|A'_{ij}| = |A_{ij}|$
 A'_{ij} 也是 S_{ij} 的一个最优解，包含 a_m

活动选择问题 (续)

3. 贪心算法

- 动态规划求解时，原问题 S_{ij} 可分解为两个子问题 S_{ik} 和 S_{kj} 求解，且这种分解有 $|S_{ij}|$ 种可能
- 定理16.1可简化问题求解过程：
 - 求 S_{ij} 最优解时只用到一个子问题，另一个子问题为空
 - 只需考虑一种选择，即选择 S_{ij} 中最早完成的活动
- 定理16.1可以自顶向下的方式解每一个子问题

活动选择问题 (续)

3. 贪心算法

- 当某个 a_m 加入解集合后，我们总是在**剩余**活动中选择**第一个不与 a_m 冲突的活动**加入解集，该活动是能够**最早完成且与 a_m 兼容的**
- 这种选择为剩余活动的调度留下了尽可能多的机会，即：留出尽可能多的时间给剩余的尚未调度的活动，以使解集合中包含的活动最多

每次选一个最早完成并与刚加入解集元素兼容的活动

活动选择问题 (续)

5. 迭代贪心算法

- RECURSIVE_ACTIVITY_SELECTOR 几乎就是尾递归：以一个对自身的递归调用再接一次并集操作结尾
- 尾递归过程改为迭代形式通常很直接，某些特定语言的编译器可以自动完成这一工作

```
GREEDY_ACTIVITY_SELECTOR(s, f)
```

```
1  n ← s.length
```

```
2  A ← {a1}
```

```
3  k ← 1
```

```
4  for m ← 2 to n do
```

```
5      if s[m] ≥ f[k]
```

```
6          A ← A ∪ {am}
```

```
7          k ← m
```

```
8  return A
```

时间复杂度： $\Theta(n)$
(已排序情况)

排序： $\Theta(n \lg n)$

算法正确性证明？
循环不变式及证明→定理16.1证明→算法正确